Computing Exact p-Value for Structured Motif

Jing Zhang[1], Xi Chen[1], and Ming Li[2]

[1] Computer Science, Tsinghua University, Beijing, 100084, China
{mitjj00,xichen00}@mails.thu.edu.cn
[2] School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada
mli@uwaterloo.ca

Abstract. Extracting motifs from a set of DNA sequences is important in computational biology. Occurrence probability is a common used statistics to evaluate the statistical significance of a motif. A main problem is how to calculate the occurrence probability of the motif on the random model of DNA sequence efficiently and accurately. In this paper, we are interested in a particular motif model which is useful in transcription process. This motif, which is called structured motif, is composed two motif words on single nucleotide alphabet and with fixed spacers between them. We present an efficient algorithm to calculate the exact occurrence probability of a structured motif on a given sequence. It is the first non-trivial algorithm to calculate the exact p-value for such kind of motifs.

Keywords: Pattern and motif discovery, exact p-value, structured motif, dynamic programming.

1 Introduction

Transcription factors play a prominent role in gene regulation; identifying and characterizing their binding sites is central to annotating genomic regulatory regions and understanding gene-regulatory networks. More and more research works focus on this field. An important aspect of this is determining the statistical significance of the occurrences of transcription factor binding site (TFBS), also called motifs, in a DNA sequence. Statistical measures used for evaluating overabundance of patterns in sequences have been studied extensively, among which the *z-score* and *p-value* are most popular. P-value is the occurrence probability of a motif on the random model of DNA sequences for at least observed times.

A widely used random model of DNA sequences is Markov chain model which considers DNA sequences as a sequence of variable indexed by a finite state Markov chain. In our paper, we use 1-order markov chain to model DNA sequences. There are many different ways to model a motif and the p-value is different under different models. The basic motif model is a word on single nucleotide alphabet $\Sigma = \{A, C, G, T\}$ or IUPAC alphabet which allows more than one nucleotide to occupy a single position. To represent more complex motifs,

B. Ma and K. Zhang (Eds.): CPM 2007, LNCS 4580, pp. 162–172, 2007.

general models such like PWM(position weight matrix) and PSSM(position specific scoring matrix) are introduced. In this representation, the nucleotide on each position of the motif is chosen from single nucleotide alphabet according to certain probabilities. A very useful model called "structured motif" is introduced by Marsan and Sagot [1,2]. The structured motif may be composed of two or more ordered of words, called "boxes". Each box is separated from the next one by a certain number of spacers("N"). The interval may be different for two pairs of consecutive boxes. We can find many motifs having the structured property in biological data such like *GAL4* which can be written as "CGGNNNNNNNNNCCG".

The non-overlapped two boxes problem has been studied by van Helden *et al.* [4] in which the structured motifs are not allowed to overlap each other when they appear. Robin S. *et al.* gave an algorithm to calculate the approximation probability of occurrence of a motif composed of two of more boxes separated by variable number of spacers in [3]. However, there is no efficient algorithm to calculate the exact probability for motif composed of exact two boxes separated by a fixed number of spacers considering motif overlap. In this paper, we propose the first non-trivial and efficient algorithm to solve such a problem. The time complexity of the algorithm is polynomial when the ratio of the number of spacers to the length of the second box is a constant.

We will give the notations and the formal description of the problem in section 2. The details of the algorithm are introduced in section 3 and we will give the conclusion and future work in section 4. We will show the details of the time complexity analysis in Appendix A. We also implement the algorithms and do some computational experiments on yeast data. The results are shown in Appendix B. The software and materials are available on our website http://bio.dlg.cn.

2 Preliminary

2.1 Basic Notations

For any string S, we use $S[i]$ to denote the ith position of S, and $S[i,j]$ to denote the substring of S from $S[i]$ to $S[j]$ inclusive, i.e. $S[i]S[i+1]\dots S[j]$. Let $\Sigma = \{A, C, G, T\}$ be the alphabet of nucleotides. The random model of DNA sequence is Markov chain with length n on Σ, i.e. we assume that it has been generated by a Markov chain. We denote by

$$m = m_1(\#N = t)m_2 \tag{1}$$

a structured motif composed of two words separated by t spacers which means that there can be any nucleotide between them. The lengths of m, m_1 and m_2 are l, l_1 and l_2 where $l = l_1 + l_2 + t$. A motif is considered to hit a string if the string contains the motif as a substring. That is, there $\exists i$, such that $m_1 = R[i, i+l_1-1]$ and $m_2 = [i+l_1+t, i+l-1]$. A_i represents an event that m hits at position i. The number of hits is the number of such different i, regardless of overlaps.

2.2 Problem Description

From a theoretical point of view, regulatory regions can be divided into two parts: the binding sites which play an important role in regulating gene expression, and the background which is not bound by transcription factors of interest. The key point for discriminating the signals from the background is to estimate if the motif is over-represented under the null hypothesis. To evaluate the statistical significance of the motif, we have to calculate the probability of a structured motif m hits a Markov region at least k times, where k is the appearance times of the motif on the given sequence. We give the formal description as following:

Input: A structured motif $m = m_1(\#N = t)m_2$ where m_1 and m_2 are motif words over alphabet $\Sigma = \{A, C, G, T\}$, an integer $k > 0$ and a Markov Region R with length n
Output: Pr(m hits region R at least k times)

3 Algorithm

To sketch the idea of the algorithm and make the description clean, we first give the algorithm for an independent and identically distributed (i.i.d) model and we will extend it to Markov model later. The main technic used here is dynamic programming. But if we use dynamic programming directly, the number of terms to calculate will expand too larger. To avoid such a problem, first we do transformations to the target probability and decompose the probability into terms which can be calculated using dynamic programming and the number of terms is constrained.

We will present the details of our algorithm for case $k = 1$, i.e the motif hits the region at least one times and extend it to general k later. When a motif m hits a region R, it can appear on any position of R, so the target probability equals to $\Pr(A_1 \bigcup A_2 \bigcup \ldots \bigcup A_n)$ where A_i represents an event that m hits at position i. We can use the principle of inclusion and exclusion to decompose the target probability. Each term in the equation is the sum of probability that the motif hits at some positions simultaneously, for example, motif m hits region at $b, b + 2$ in Figure 2 at the same time. We further decompose the term by the first hit position and second hit position. After decomposition, the term is the probability that the motif hits a shorter region with a well-defined prefix for fewer times and we can use dynamic programming to calculate it.

3.1 Decomposition and Transformation

First, We use the principle of inclusion and exclusion to decompose the target probability and classify the hit events by the first hit position.

$$\begin{aligned} \Pr(m \text{ hits } R) &= \Pr(A_1 \bigcup A_2 \bigcup \ldots \bigcup A_n) \\ &= \sum_{a=1}^{n} (-1)^{a-1} \sum_{1 \le i_1 < i_2 < \ldots < i_a \le n} \Pr(\bigcap_{v=1}^{a} A_{i_v}) \end{aligned} \tag{2}$$

$$= \sum_{a=1}^{n} (-1)^{a-1} \sum_{b=1}^{n} \sum_{i_1=b<i_2<\ldots<i_a\leq n} \Pr(\bigcap_{v=1}^{a} A_{i_v}) \tag{3}$$

The equation (2) is from the principle of inclusion and exclusion . We classify the events according to different value of i_1 to get the equation (3). Let

$$P(a,b) = \sum_{i_1=b<i_2<\ldots<i_a\leq n} \Pr(\bigcap_{v=1}^{a} A_{i_v}) \tag{4}$$

The key problem is how to calculate $P(a,b)$ for all a and b from 1 to n.

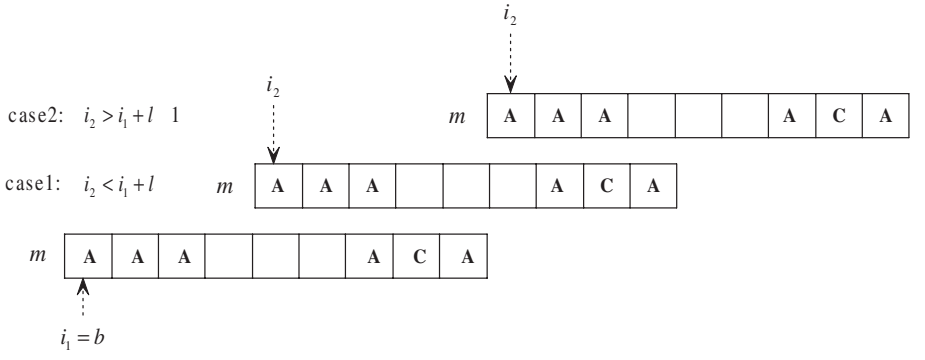

Fig. 1. Two cases of i_2

We do further decomposition to $P(a,b)$. When i_1 decided, there are two kinds of relation between i_1 and i_2, one is that the motif starting from i_1 does not overlap that starting from i_2 and the other is overlap case. We classify the events by different value of i_2 in equation (5):

$$P(a,b) = \Pr(A_b) \times \sum_{b_1=b+l}^{n} P(a-1,b_1) \tag{5}$$
$$+ \sum_{b_2=b+1}^{b+l-1} \sum_{i_1=b<i_2=b_2<\ldots<i_a<n} \Pr(\bigcap_{v=1}^{a} A_{i_v})$$

Let

$$P_1(a,b) = \sum_{b_2=b+1}^{b+l-1} \sum_{i_1=b<i_2=b_2<\ldots<i_a<n} \Pr(\bigcap_{v=1}^{a} A_{i_v}) \tag{6}$$

$$P_2(a,b) = \Pr(A_b) \times \sum_{b_1=b+l}^{n} P(a-1,b_1)$$

We can see that P_1 corresponds to the overlap case and P_2 the non-overlap case. P_2 can be calculated by $P(a-1, b_1)$ which is already known in dynamic programming, but we can not calculate P_1 using dynamic programming directly and we keep on doing transformations to P_1.

In fact, the events A_{i_1} and A_{i_2} can not hold simultaneously for many choices of i_2. For two given motif m' and m'', we define the **compatible** position set.

Definition 1. *For given motif m' and m'' with length l on alphabet $\{A, C, G, T, N\}$. Let $Q(m', m'')$ denote the **compatible** position set of m' and m''. For any i that $0 < i \leq l$, $i \in Q(m', m'') \iff \forall 1 \leq j < i, m'[j] = m''[l-i+j]$ or $m'[j] = N$ or $m''[l-i+j] = N$.*

In another word, $Q(m', m'')$ are all the possible positions for m'' to appear in when there is already a motif m' appears. We constrain the choices of i_2 in equation (6) and rewrite it.

$$\begin{aligned} P_1(a,b) &= \sum_{d \in Q(m,m)} \sum_{i_1=b<i_2=b+d<\ldots<i_a \leq n} \Pr(\bigcap_{v=1}^{a} A_{i_v}) \\ &= \sum_{d \in Q(m,m)} \Pr(m[1,d] = R[b, b+d-1]) \times \sum_{b+d<i_3<\ldots<i_a \leq n} \\ &\quad \Pr(s_d \text{ is the prefix of } R[b+d, n], \bigcap_{v=3}^{a} A_{i_v}) \end{aligned} \tag{7}$$

in which $\Pr(m[1,d] = R[b, b+d-1])$ can be calculated using the parameters of i.i.d model directly. We show the meaning of s_d in Figure 2 for case $d = 9$. The basic idea of equation (7) is to divide the region $R[b, n]$ into two parts at i_2. A suffix of the motif m which starts from $i_1 = b$ locates in region $R[b+d, n]$. The combination of the suffix and the motif m which starts from $i_2 = b + d$ forms the prefix s_d.

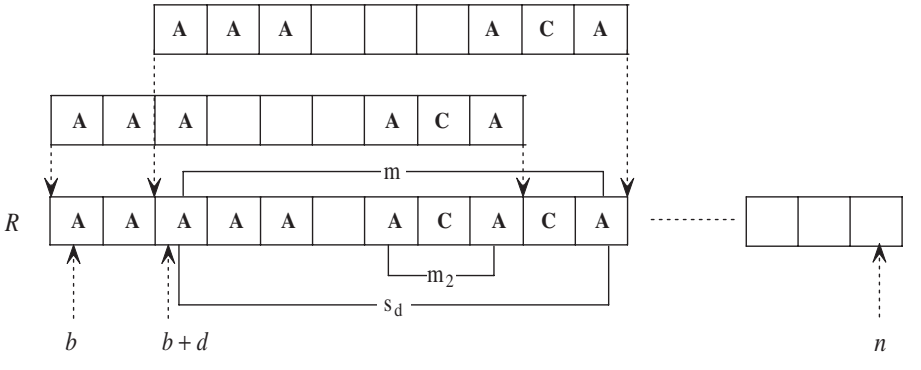

Fig. 2. Cut the region from $b + d$ and form the prefix s_d

Till now, we keep on doing transformations to the target probability and try to find some variants which are easy to calculate using dynamic programming and the number of variants does not expand too fast during the recursion. We will define another probability in subsection 3.2 and show that $P(a, b)$ and the well-defined probability can be calculated by dynamic programming together.

3.2 Dynamic Programming

The probability $I(x, y, z)$ is defined to be:

$$I(x, y, z) = \sum_{y \le i_1 < i_2 < \ldots < i_x \le n} \Pr(z \text{ is a prefix of } R[y, n], \bigcap_{v=1}^{x} A_{i_v}) \tag{8}$$

in which $1 \le x, y \le n$ and $s \in SP(m)$. $SP(m)$ is the structured prefix set of m which is defined as following:

Definition 2. *A string s with length l belongs to the **structured prefix set** of a given motif $m = m_1(\#N = t)m_2$ when it satisfies the three constraints:*
Constraint 1: $s[1, l_1] = m_1$ and $s[l - l_2 + 1, l] = m_2$
Constraint 2: $s[l - d - l_2 + 1, l - d] = m_2$, where $d \in Q(m, m)$.
Constraint 3: Other positions are covered by spacers or several overlapped m_2.

We can see that when the other positions are only covered by spacers, it is exactly the prefix s_d. Assume the number of m_2 used to do covering is r, we illustrate all the possible structured prefixes when $m = AAA(\#N = 5)ACA$ and $d = 9$ in Figure 3. We can regard it as using several m_2 to cover a string with length t and filling the rests with spacers. It is possible that several m_2 overlap each other and overlap $s[1, l_1]$ or $s[l - l_2 + 1, l]$. The constraint 3 is meant to make the prefix set complete in dynamic programming.

Then we prove that $P(a, b)$ and $I(x, y, z)$ can be calculated by dynamic programming. Since the complexity is related to the size of $SP(m)$, we just use $|SP(m)|$ to represent the size of $SP(m)$ in the time complexity analysis and the details of the estimation to $|SP(m)|$ are given in Appendix A.

Lemma 1. *For $1 \le x, y \le n$ and $z \in SP(m)$, $1 \le a, b \le n$, all $I(x, y, z)$ and $P(a, b)$ can be calculated using Dynamic Programming. If the size of $SP(m)$ is $|SP(m)|$, the total time complexity is $O(n^3|SP(m)|)$.*

Proof. It is easy to see that $P_1(a, b) = \sum_{d \in Q(m,m)} \Pr(m[1, d] = R[b, b + d - 1]) \times I(a - 2, b + d, z_d)$, so $P(a, b)$ can be calculated by $P(a', b')$ and $I(x, y, z)$ already known in dynamic programming. We calculate $I(x, y, z)$ for y from n to 1, x from 1 to $n - y$ and z in the structured prefix set of m in arbitrary order. When $x = 0$, $I(x, y, z) = \Pr(z \text{ is a prefix of } R[y, n])$ is easy to calculate. Assume we have got the value of $I(x', y', z)$ for all $y' > y$, $x' < x$, any $z \in SP(m)$. We do decomposition to $I(x, y, z)$ according to the first hit position:

$$I(x,y,z) = \sum_{b=y}^{n} \sum_{i_1=b<i_2<\ldots<i_x<n} \Pr(z \text{ is a prefix of } R[y,n], \bigcap_{v=1}^{x} A_{i_v}) \tag{9}$$

$$= \sum_{b_1=y+l}^{n} (\Pr(z \text{ is a prefix of } R[y,n]) \times P(x,b_1)) \tag{10}$$

$$+ \sum_{b_2-y \in Q(z,m)} (\Pr(z[1,b_2-y+1] = R[y,b_2]) \times I(x-1,b_2,z')$$

Since z' is a string covered by the prefix m and $z[b_2-y,l]$, it is still in $SP(m)$. The number of different $I(x,y,z)$ is $n^2|SP(m)|$ and the time to calculate each $I(x,y,z)$ is $O(n)$, so the total complexity is $O(n^3|SP(m)|)$.

We conclude the recursion formula for $I(x,y,z)$ and $P(a,b)$ as following:

$$P(a,b) = \Pr(A_b) \times \sum_{b_1=b+l}^{n} P(a-1,b_1)$$

$$+ \sum_{d \in Q(m,m)} \Pr(m[1,d] = R[b,b+d-1]) \times I(a-2,b+d,z_d)$$

$$I(x,y,z) = \sum_{b_1=y+l}^{n} (\Pr(z \text{ is a prefix of } R[y,n]) \times P(x,b_1))$$

$$+ \sum_{b_2-y \in Q(z,m)} (\Pr(z[1,b_2-y+1] = R[y,b_2]) \times I(x-1,b_2,z')$$

We can see that the DP algorithms to calculate $P(a,b)$ and $I(x,y,z)$ get involved with each other. □

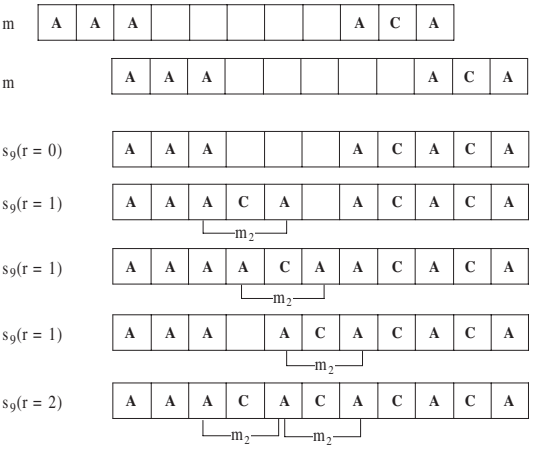

Fig. 3. all the prefixes in the structured prefix set when $m = AAA(\#N = 5)ACA$ and $d = 9$. r is the number of m_2 to do covering in constraint 3

Given all $P(a,b)$ for $1 \le a,b \le n$, we can calculate the target probability by

$$\Pr(m \text{ hits region } R \text{ at least } k \text{ times}) = \sum_{a=k}^{n} (-1)^{(a-k)\%2} \sum_{b=1}^{n} P(a,b) \qquad (11)$$

In Appendix A, we prove that the size of the structured prefix set of a structured motif $m = m_1(\#N = t)m_2$ is $O((2l_2+2)^{2t/l_2+2})$. When t/l_2 is a constant, it is $O(l_2^c)$ for some constant c. We have the following theorem:

Theorem 1. *Given a structured motif $m = m_1(\#N = t)m_2$ on alphabet $\Sigma = \{A,C,G,T\}$ and an i.i.d region R of length n, we can calculate* $\Pr(m$ *hits region R at least k times) in $O(n^3 \times s_{max} + t_{max} \times r_{max})$ where $s_{max} = (2l_2+2)^{2t/l_2+2}$ and $r_{max} = 2 \times \frac{l_2+t}{l_2+1}$. When t/l_2 is some constant c, the time complexity is $O(n^3 \times l_2^c)$.*

3.3 Sketch of Algorithm on Markov Model

The algorithm on markov model is quite similar to that on i.i.d model. The key idea for the extension is to maintain the last char before the region. We can rewrite the Subsection 3.1 and 3.2 using this idea and the frame of the algorithm and time complexity analysis remains the same.

We take equation (4)∼(7) in decomposition and transformation as an example do show the extension. We rewrite equation (4) as

$$P_M(a,b,c) = \sum_{c \in \Sigma} \sum_{i_1 = b < i_2 < \ldots < i_a \le n} \Pr(\bigcap_{v=1}^{a} A_{i_v} | R[b-1] = c)$$

which is the sum over conditional probabilities with different char before region $R[b,n]$. We can also classify the events by different value of i_2 similar to equation (5) as following:

$$\begin{aligned} P_M(a,b,c) = \sum_{c \in \Sigma} (\Pr(A_b | R[b-1] = c) \sum_{d \in \Sigma} \sum_{b_1 = b+l}^{n} (P(a-1, b_1, d) \times \Pr(R[b_1 - 1] \\ = d | A_b)) + \sum_{b_2 = b+1}^{b+l-1} \sum_{i_1 = b < i_2 = b_2 < \ldots < i_a < n} \Pr(\bigcap_{v=1}^{a} A_{i_v} | R[b-1] = c)) \end{aligned}$$

The conditional probability in the non-overlap part P_{M_2} can be calculated using the transition probabilities of markov model. For the overlap part P_{M_1}, we rewrite it like equation (7):

$$\begin{aligned} P_{M_1} = \sum_{c \in \Sigma} (\sum_{d \in Q(m,m)} \Pr(m[1,d] = R[b, b+d-1] | R[b-1] = c) \times \sum_{b+d < i_3 < \ldots < i_a \le n} \\ \Pr(s_d \text{ is the prefix of } R[b+d, n], \bigcap_{v=3}^{a} A_{i_v} | R[b+d-1] = m[d])) \end{aligned}$$

The extension in Dynamic Programming is quite similar to that of decomposition and transformation part.

4 Conclusion and Future Work

In this paper, we present a non-trivial and efficient algorithm to calculate the probability of the occurrence of a structured motif $m = m_1(\#N = t)m_2$ where m_1 and m_2 are motif words on basic alphabet $\Sigma = \{A, C, G, T\}$ with t spacers. We do transformations to the target probability and define two variants which can be calculated by DP algorithm. The time complexity of the algorithm is $O(n^3 \times l_2^c)$ for some constant c when t/l_2 is a constant where l_1 and l_2 are the length of m_1 and m_2.

The algorithm can be used to evaluate the statistical significance of motif candidates with structured property. This kind of motifs are often with larger length in real biological data and it is harder to calculate the occurrence probability. Our algorithm is the first non-trivial and efficient algorithm to calculate the exact p-value of such kind of motifs.

One problem is that the algorithm is still an exponential time algorithm in worst case. Finding a polynomial time algorithm or proving that it is NP-hard are two main directions in the future work.

Acknowledgement

ZJ' and CX's work is supported by the National Natural Science Foundation of China Grant 60553001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901. ML's work was partially supported by the Chang Jiang Scholarship Program, NSERC, and Canada Research Chair program.

References

1. Marsan, L., Sagot, M.F.: Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification. J. Comp. Biol. 7, 345–362
2. Marsan, L., Sagot, M.F: Extracting structured motifs using a suffix tree-algorithm and application to promoter consensus identification. In: RECOMB'00 Proceedings of Fourth Annual International Conference on Computational Molecular Biology, pp. 210–219. ACM Press, New York (2000)
3. Robin, S., Daudin, J.-J., Richard, H., Sagot, M.-F., Schbath, S.: Occurrence probability of structured motifs in random sequences. J. Comp. Biol. 9, 761–773 (2002)
4. Van Helden, J., Rios, A.F., Collado-Vides, J.: Discovering and Regulatory elements in non-coding sequences by analysis of spaced dyads. Nucl. Acids Res. 28, 1808–1818
5. Zhu, J., Zhang, M.Q.: SCPD: A promoter database of yearst saccharomyces cerevisiae. Bioinformatics 15, 607–611 (1999)

A Estimation of the Size of Structured Prefix Set

The following lemma is about the size of the structured prefix set of m.

Lemma 2. *The size of the structured prefix set of a structured motif* $m = m_1(\#N = t)m_2$ *is* $O((2l_2 + 2)^{2t/l_2+2})$. *When* t/l_2 *is a constant, it is* $O(l_2^c)$ *for some constant* c.

Proof. According to definition 2, for any string s in $SP(m)$, the contents of the first l_1 and the last l_2 length of s are fixed. Although there must be a m appeared at some position in constraint 2, this constraint is just for the convenience to define s_d from $SP(m)$ and it has been contained in constraint 3. We can omit it when estimating the size of $SP(m)$. The number of different $s[l_1+1, l_1+t]$ is exact the size of $SP(m)$, so we prove that the number of different s^* is at most $O((2l_2+2)^{2t/l_2+2})$ where $s^* = s[l_1+1, l_1+t]$.

First, we give the upper bound of the needed number of m_2 to cover the whole s^*. Assume we use $m_2^{(1)}, m_2^{(2)}, \ldots, m_2^{(t)}$ to cover s^*. $m_2^{(j_1)}$ appears before $m_2^{(j_2)}$ if $j_1 < j_2$. Make sure that every $m_2^{(j)}$ is needed. Here, the needed means that there exists a substring of s^* which is only covered by $m_2^{(j)}$. We have the following to facts:

Fact 1. *$\forall 1 \le j \le r$, $m_2^{(j)}$ covers at a substring of s^* with length at least* 1.

Fact 2. *$m_2^{(j)}$ can not overlap $m_2^{(j+2)}$.*

Fact 2 is right because if they overlap each other, $m_2^{(j+1)}$ is not needed. Combine the two facts, we have an upper bound of r,

$$r \le 2 \times \frac{l_2+t}{l_2+1} = r_{max}$$

The number of different s^* is bounded by the number of choosing r positions from l_2+t possible start positions for m_2 in which r is ranging from 0 to r_{max}.

$$\begin{aligned} \text{number of different } s^* &\le \sum_{r=0}^{r_{max}} \binom{l_2+t}{r} \le \left(\frac{l_2+t}{r_{max}}\right)^{r_{max}} \\ &= (2l_2+2)^{2\times\frac{l_2+t}{l_2+1}} \le (2l_2+2)^{2t/l_2+2} \end{aligned}$$

Let s_{max} denote $(2l_2+2)^{2v/l_2+2}$. We can construct the structured prefix set just by trying all the possible choices of r from 0 to r_{max} increasingly and the total construction time complexity is bounded by $O(r_{max} \times s_{max})$.

B Computational Experiment

We implement the algorithm in C++ and do computational experiment on yeast data. GAL4 is a family of transcription activator of genes. It has a typical Zn-binding region and is reported to bind to a 17 base-pair palindromic site. The consensus binding motif is "CGGNNNNNNNNNNCCG" which can be regarded as a structured motif of two boxes with length 3 and 11 spacers between them.

In *Saccharomyces crevisiae* Promoter Database (SCPD) [5], transcription factor *GAL4* is reported to bind to 7 Genes: GAL1, GAL2, GAL4, GAL7, GAL10, GAL80 and GCY1. We extract upstream sequence, of length 1000 bp, for these 7 genes and estimate the occurrence probabilities of $\{A, C, G, T\}$ on each sequence.

We calculate the exact p-value of motif *GAL4* to appear on each sequence for observed times. The p-value and consumed time[1] are shown in Table 1. If we use $\alpha = 0.05$ in hypothesis test to decide whether motif GAL4 is over-represented in the promoter region, we can see that all the p-values are significant below α except GAL80. The result shows the power of the algorithm to discriminate the transcription signals from the background. The average time is $12.6s$ which is acceptable in application. We also enumerate all the structured motifs with

Table 1. The p-value of motif *GAL4* on corresponding Genes

Gene Name	P-value of motif GAL4	Time
GAL1	9.61361E-06	12813ms
GAL2	2.16228E-09	12734ms
GAL4	0.006578594	12609ms
GAL7	0.034349462	12532ms
GAL10	9.61361E-06	12641ms
GAL80	0.092351709	12282ms
GCY1	0.045182862	12640ms

$l_1 = l_2 = 3$ and $t = 11$. The number of such motifs is $4^6 = 4096$. We calculate the exact p-value of all such motifs on Gene GAL4 and rank them according to p-value increasingly. The top 20 p-values are shown in Table 2. We can see that the rank of "CGG(#N=11)CCG" is the 11-th of all 4096 structured motifs of similar structure, that is in the top 0.2%.

Table 2. The top 20 p-value of structured motif on Genes *GAL4*

Structured Moitf	P-value	Structured Moitf	P-value
TTT(#N=11)TTT	0.00135	CGG(#N=11)CCG	0.00658
ACA(#N=11)AGG	0.00141	AAA(#N=11)CTT	0.00689
AGA(#N=11)CAG	0.00151	ATT(#N=11)ACA	0.00704
GTG(#N=11)AGA	0.00163	GGA(#N=11)GGC	0.00858
AGC(#N=11)TCA	0.00182	CGG(#N=11)AGG	0.00858
GAC(#N=11)CTA	0.00183	TTA(#N=11)TTC	0.00963
TTT(#N=11)CGC	0.00273	GGT(#N=11)CGG	0.01004
ATT(#N=11)GTG	0.00393	GGG(#N=11)TCC	0.01020
TTT(#N=11)GAG	0.00430	GTC(#N=11)CGG	0.01037
CAC(#N=11)TTT	0.00516	TTT(#N=11)CTT	0.01238

[1] The compiler is Microsoft VC 6.0 and The test PC is equipped with a Pentium 4 running at 2.8GHz and 512MB of RAM.